

104230" 00704660

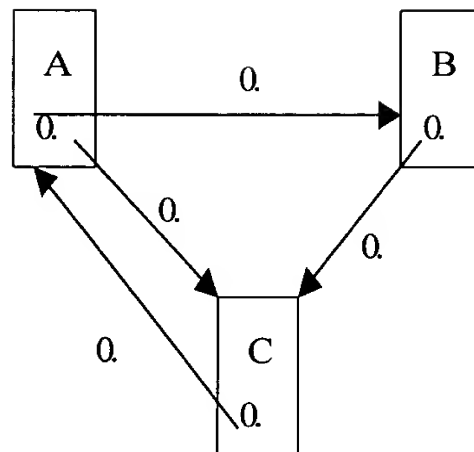


Figure 1

```
graph TD; A[URL-QUEUE of all uncrawled pages] --> B([Parse one page and extract links]); A --> C[Pool of CRAWLED_PAGES]; B --> D([Fetch new pages the links point to]); B --> E[List of LINKS]; D --> F[URL_QUEUE of all uncrawled pages]; E --> G[Compute ranks for all pages]; F --> G; H[Pool of CRAWLED_PAGES] --> G; G --> A;
```

The flowchart illustrates the iterative PageRank algorithm. It begins with a **URL-QUEUE of all uncrawled pages**. The process involves parsing one page and extracting links, which are then used to fetch new pages. The extracted links are stored in a **List of LINKS**, and the crawled pages are added to a **Pool of CRAWLED_PAGES**. The process then computes ranks for all pages, which are used to update the **URL_QUEUE of all uncrawled pages** for the next iteration.

Figure 2

```

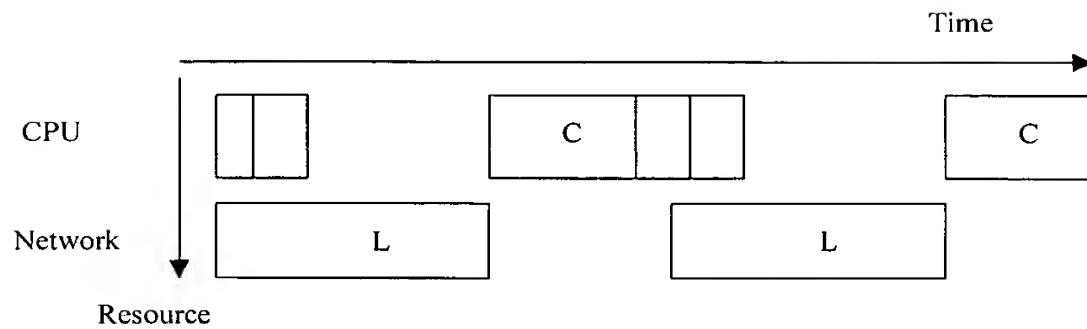
Enqueue(URL_QUEUE, starting_url);

While (! Empty(URL_QUEUE)) {
    url = Dequeue(URL_QUEUE);
    page = Crawl_page(url);
    Enqueue(CRAWLED_PAGES, (url, page));
    PAGE_TO_FETCH_LIST = Extract_urls(page);
    For each u in PAGE_TO_FETCH_LIST {
        Enqueue(LINKS, (url, u));
        If ((u not in URL_QUEUE) and ((u, -) not in CRAWLED_PAGES)) {
            Fetch_page(u);
            Enqueue(URL_QUEUE, u);
        }
    }
    Compute_PageRank(URL_QUEUE+CRAWLED_PAGES);
    Reorder_queue(URL_QUEUE);
}

```

Figure 3

10.230.337.0-1660



F: Initiating page fetches (protocol management)
 R: Receiving/taking pages from incoming queue
 C: Computation
 L: Loading pages from remote servers through network

Figure 4

10/22/2010 10:40:10 AM

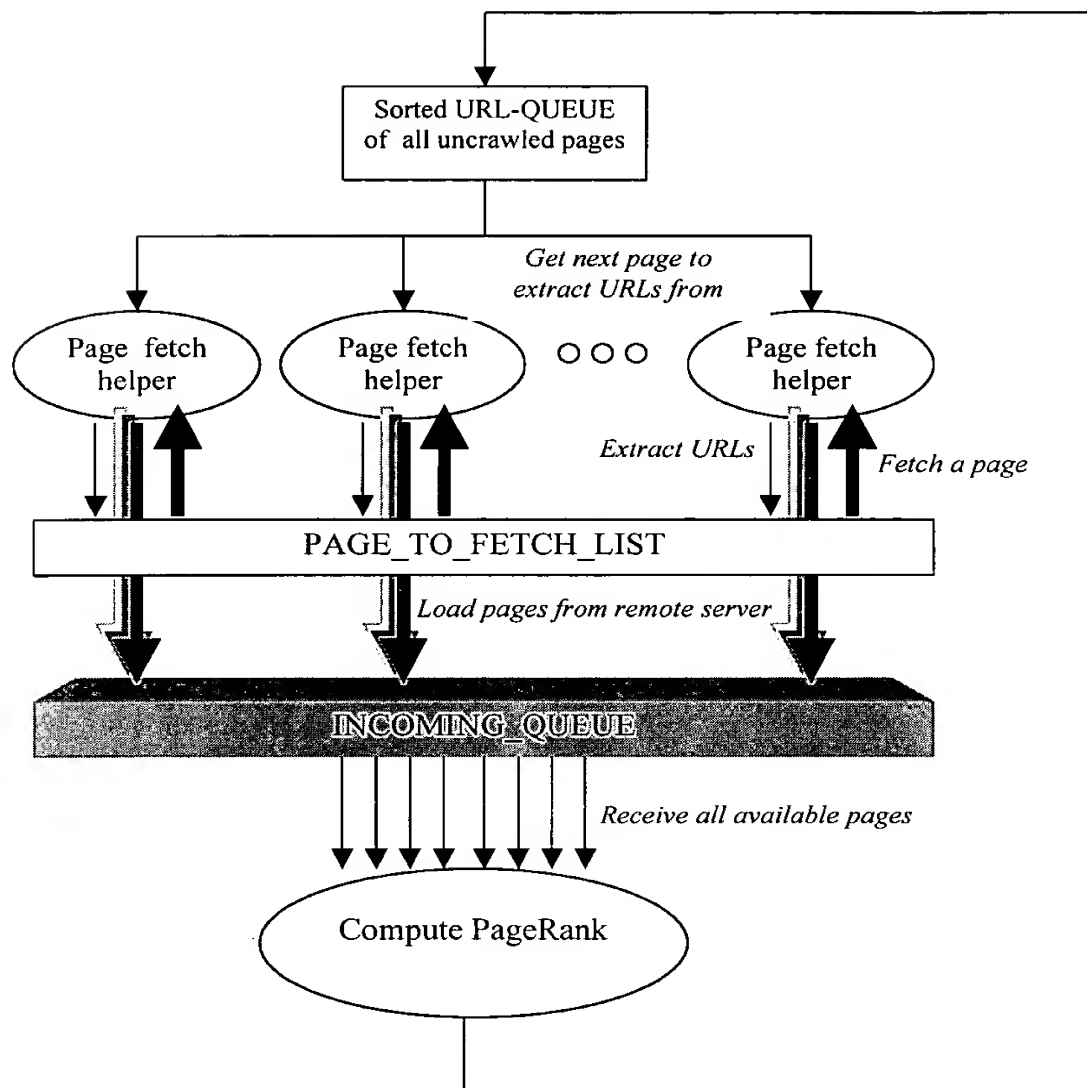


Figure 5